F O X T R A I L S

A Web Development Framework for Foxpro

TUTORIAL

PHOTO COURTESY OF

Pennsylvania Game Commission

About Fox Trails

Once upon a time...

When I was a young lad, getting started in my programming career, I had the good fortune to find a copy of Yair Alan Griver's Codebook. This book was very influential in shaping my development practices for more than a decade. Those who have read this work may know why I hold it in high regard. Those who have not will doubtless see some of his concepts presented as we go through this tutorial.

A simpler approach

Web development may appear to be quite simple. Indeed, for static sites, a word processor may be all that is needed to produce a stunning site. However, as sites become more complex, or require integration with back-end databases, the complexity of such sites can soon get overwhelming.

In 2006, I was asked to design an AJAX-enabled web site for QuotePro, Inc. The web application needed to integrate with a mission critical insurance application written in Foxpro. The design called for use of a VFP COM server to expose the business logic for use by ASP.Net and the Prototype AJAX library. At about the same time, I began experimenting with Ruby On Rails during my spare time.

Every new function that I added to the VFP Com server required a time-consuming process:

1. Close Visual Studio;
2. Recompile the dll in VFP;
3. Restart Visual Studio;
4. Recreate the ASP.Net Interop dll:
5. Test everything;
6. Stop the production web server;
7. Install all the updated files
8. Restart the server.

It was tedious compared with the instantaneous updates that Ruby on Rails exhibited.

I began looking for a simpler approach, a way to integrate a Rails-like approach with a Visual Foxpro backend. I was hoping to gain some of the same efficiencies of Rails in a VFP based web framework.

Fox Trails design goals

The new framework would need to achieve the following design goals for it to be successful:

1. Easy to use framework - no deeply nested object graphs.
2. Simple configuration - use meta data where available to eliminate configuration issues.
3. Easy to deploy - separate application logic from the COM interface layer.
4. Freely available - Fox Trails is public domain software.

The Dispatcher

Fox Trails contains only one OLEPUBLIC class. This class acts as a translator between ASP.Net and Foxpro. To use terminology from Codebook, it is a traffic cop. New features can be installed on a production server by simply copying new Foxpro programs to the server.

We'll discuss the dispatcher in more detail later. For now, just remember that this is where all the magic happens. Let's get the Fox Trails demo application installed and running:

Getting Started

Obtaining Fox Trails

Fox Trails may be downloaded from http://www.marquisenterprises.biz/wsvn/FoxTrails, choose the tarball next to the FoxTrails trunk. You may also download Fox Trails using a subversion client from http://www.marquisenterprises.biz/svn/FoxTrails/trunk

Extract the tarball to the location of your choice. For simplicity, we'll assume it's installed in C:\foxtrails.

A Few Prerequisites

You'll need a copy of Visual Foxpro 9.0 installed (older versions may work, but are untested).

You'll need to install Internet Information Services on your computer and then the .Net 2.0 framework. Note that if you install .Net 2.0 before IIS, you'll need to reinstall it after the IIS installation has completed.

Although it's not required, Visual Studio 2005 will be helpful for debugging Fox Trails applications.

Configuration Steps

1. Open c:\foxtrails\demo.pjx and build a multi-threaded dll (or choose Start/Run, regsvr32 c:/foxtrails/demo.dll).
2. If you have Visual Studio 2005, double-click the c:\foxtrails\webroot\webroot.sln file.
3. Open the default.aspx file in Visual Studio and press F5.

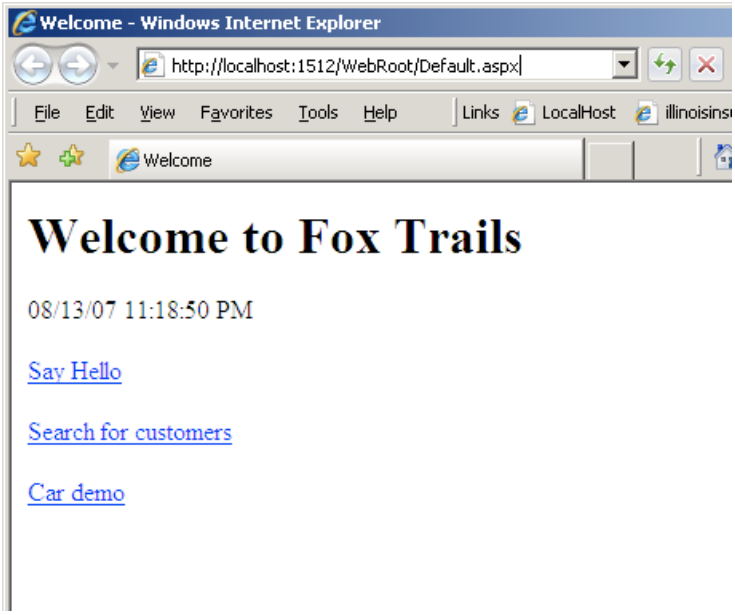
If you don't have Visual Studio, you'll need to perform the following additional steps:

4. create a virtual directory under IIS and point it to c:\foxtrails\webroot. You can name the virtual directory whatever you want (or make WebRoot the site's home directory in IIS). We'll assume a virtual directory named WebRoot.
5. Verify that the ASP.Net user has read access to c:\foxtrails and it's sudirectories.
6. Add modify permissions on the Db subdirectory. (I usually just set Everyone permissions on development systems).
7. Open a web browser and enter http://localhost/webroot

Welcome to Fox Trails

Fox Trails Conventions

If all went well in the preceding steps, you should now see a page similar to the image on the right. If not, drop me an email and I'll do what I can to help. bmarquis65@mac.com.



Move your mouse over the links on the page and notice the pattern used to name links in Fox Trails. This is how Fox Trails performs it's magic. Through the use of naming conventions, we can identify which FoxPro class to load and what method to call on that class when a link is clicked.

Here is the pattern:

http://localhost/webroot/default.aspx/classname/methodname/parameter1/parameter2/...

The dispatcher class transforms this web address, aka a URL, into the following commands:

instance = CREATEOBJECT("classname")
instance.method(parameter1,parameter2)

More about naming conventions

Traditional application development, especially web development, requires numerous configuration files or tables so that the application can find it's data files, determine whether it's running in development mode or deployment, defines enabled features, user access rights and so on.

Codebook was revolutionary, at least for me, in that it introduced the concept of naming conventions in Foxpro development. Window names corresponded to the screen set program (.SPR) needed to display that window. The Traficop served as the foundation read, the predecessor of today's READ EVENTS clause. It coordinated all of the actions a user could perform in an application. Variable names likewise had naming conventions to relay the fields scope and type. Variable naming conventions are often referred to as Hungarian Notation.

Ruby on Rails takes the concept of naming conventions one step farther, to provide applications with "self-awareness". An example will help clarify how this is done. RoR applications are web applications, so let's start with a typical URL.

http://localhost/RoR/customer/list

From this Url, RoR immediately "knows" it needs a customer controller, a customers table (or model) and an html template named list.rb. After passing the list message to the customer controller, the list.rb template is rendered and the resulting html is displayed. All of this logic happens behind the scenes, allowing the developer to concentrate on

writing their application instead of wasting time configuring the application. RoR also uses database meta data to avoid hard coding field names in the application. A full discussion of RoR is beyond the scope of this tutorial. If you are interested in more details on RoR see:
http://www.onlamp.com/pub/a/onlamp/2006/12/14/revisiting-ruby-on-rails-revisited.html

Fox Trails uses these concepts to provide many of these same features to Foxpro web development.

Say Hello

Following in the tradition of Kernigan and Ritchie's classic "Hello World" application, let's start our tutorial with a simple web page.

Modify Command Controllers\say.prg

Modify the hello procedure to check for the existence of a parameter:

```
PROCEDURE hello
LPARAMETERS lcUser
IF PCOUNT() = 0
        lcUser = "World"
ENDIF
this.greeting = "Hello " + lcUser
ENDPROC
```

Run default.aspx from Visual Studio

and call the hello method

http://localhost:1234/webroot/default.aspx/say/hello, replacing 1234 with the actual port number you are using.

You can pass a parameter to the say method by adding another slash following hello. For example:

http://localhost:1234/webroot/default.aspx/say/hello/Cool+World!

Open up the C:\foxtrails\Views\say\hello.vml file in your favorite editor. You should see a file similar to:

```
<html>
<head>
        <title>Fox Trails</title>
</head>
<body>
<<say.greeting>>
</body>
</html
```

If you've worked with FoxPro's text merge commands, you'll notice the familiar textmerge delimiters "<<" and ">>". FoxTrails comes with a templating system which allows you to generate html without embedding html formatted strings throughout your programs (See Utility\rendervml.prg). Contrast the above hello.vml file with the equivalent Foxpro code:

```
#define cLf CHR(10)
LOCAL lcHtml
lcHtml = "<html>"+cLf +"<head>"+cLf+"<title>FoxTrails</title>"+cLf+"</head>"+cLf+"<body>"+cLf
+this.greeting+cLf+"</body>"+cLf+"</html>"
return lcHtml
```

Of course, this could be replaced by a TEXT TO statement

```
LOCAL lcHtml
TEXT TO lcHtml TEXTMERGE
<html>
<head>
        <title>Fox Trails</title>
</head>
<body>
<<this.greeting>>
</body>
</html
ENDTEXT
return lcHtml
```

While this example is much more readable, it still embeds html into your program. Foxpro is a wonderful programming tool, but it's not a great Html editor. Moreover, many web pages are developed by a team of individuals including web/graphic designers, programmers, database administrators and networking specialists. Moving the presentation into separate files enables greater collaboration among these different disciplines.

Debugging

Open Foxpro and cd to C:\foxtrails
Run debug.aspx from Visual Studio.
You should see the Foxpro debugger window open.
Now you can step into Foxpro to examine the inner workings of the FoxTrails dispatcher.

To step into the say hello method, use a URL similar to the following:

http://localhost:1234/webroot/debug.aspx/say/hello

Scaffolding

Just as contractors use scaffolds to help them build, RoR and Fox Trails both provide a utility to create a "scaffold". A scaffold is a generic maintenance set, with the ability to Create, Retrieve, Update and Delete records from a table.

In order to create a scaffold, we will first need a table. In this case, let's make an employee table in Foxpro:

```
CD C:\Foxtrails
CREATE TABLE DB\Employees ( EmployeeId C(10), FullName C(30), HireDate D, Salary N(6,0) )
INDEX ON employeeid TAG employeeid
USE
DO utility\scaffold
```

When finished, you should have several new files
Controllers\employee.prg
Layouts\employee.vml
Views\employee*.vml

Take a few moments to examine these files, then step into them using the Visual Studio debugger.

http://localhost:1234/webroot/debug.aspx/employee

Try adding a few records and see how the dispatcher acts like a traficop directing the user's request to the correct Foxpro program. Astute readers may notice that a url may have problems passing a date to Fox Trails.

http://localhost:1234/webroot/debug.aspx/appointment/add/09/07/2007

There are a few ways to handle this. Perhaps the easiest is to simply use three parameters in your code. You can also use a query string or an http post to send date fields to Fox Trails.

The generated scaffold makes a few assumptions about the type and location of the fields in the table. The first field is assumed to be a primary key and the second field is assumed to be a description or title for the record. Both are assumed to be character fields. Of course, you can easily change the generated files as needed.

VML Templates

You've already seen several examples of vml templates at this point. VML templates provide support for the default text merge delimiters "<<" and ">>". Fox Trails also supports Embedded Foxpro commands within VML files using "<%" and "%>". This was done to provide flow control and repetition within the templates. Although you can write entire programs using VML templates, doing so would violate a principle of MVC programming, embedding controller logic within the presentation layer. It is recommended that Foxpro commands within VML templates be limited to flow control and formatting. Consider creating utility functions if you need more elaborate logic in your templates. This will allow you to keep your logic separated from it's display.

Here's an example that simply counts to 10.

```
<%
LOCAL lnCount
FOR lnCount = 1 TO 10
%>
```

```
<< lnCount >>
<% ENDFOR %>
```

You can also use Foxpro commands inline: <% SCAN %><< customer.firstname >><% ENDSCAN %>

Here is a more elaborate example for embedding a standard header and footer in a VML template:

File: Layouts\Header.vml

```
<html>
<head>
<title>Some Title</title>
</head>
<body>
<% lcNavigation = RenderVml("Layouts\navigation.vml", oDispatchContext.Instance) %>
<< lcNavigation >>
```

File: Layouts\Footer.vml

```
<hr/>
<center><b>Powered by Fox Trails</b></center>
</body>
</html>
```

File: Layouts\Cart.vml

```
<% lcHeader = RenderVml("Layouts\header.vml", oDispatchContext.Instance) %>
<<lcHeader>>
<form action="<<fullurl("/cart/checkout")>>" method="post">
<fieldset><legend>Shopping Cart</legend>
<table>
<tr><th>Product</th><th>Qty</th><th> </th></tr>
<% SELECT carts
    SCAN FOR sid = oDispatchContext.Instance.sid
%>
<tr><td><input type="hidden" name="seq" value="<<carts.seq>>"/>
        <<carts.product>>
    </td>
    <td><input type="text" name="_nqty<<carts.seq>>" value="<<carts.qty>>" />
    </td>
</tr>
<% ENDSCAN %>
</fieldset>
</form>
<% lcFooter = RenderVml("Layouys\footer.vml", oDispatchContext.Instance) %>
```

<< lcFooter >>

As with the example above, you may find it necessary to embed templates inside of other templates. When Fox Trails was first released, embedded templates were not possible due to limitations with Foxpro's textmerge capabilities. A new TEXTMERGE() function helps, but will not execute embedded foxpro commands. The workaround is to render the embedded template between <% %> tags as is done in the example above. You can then display the result using subsequent textmerge delimiters << >>.

Form variables

When submitting a form from a web page, Fox Trails will add the form fields as properties of the controller object. All form fields will transmit their values as character data back to Fox Trails, regardless of the underlying data type. Fox Trails provides a way to automatically translate the incoming data into the correct type so it can be stored into the database table. Naming conventions are used to inform Fox Trails about the data type of the fields. If you precede the field name with an underscore and a single character code corresponding to a standard Foxpro Type, it will be converted into the appropriate type automatically. To store the incoming values into the database you can use a method like this:

```
PROCEDURE save
IF SEEK( this.customerid)
        GATHER this
ELSE
        INSERT INTO customers FROM this
ENDIF
ENDPROC
```

Object lifetimes

The current state of all web application development is based upon a concept known as stateless programming. When a client requests a page from a web server, the web server looks up the appropriate page and then returns the results to the client. Once the client retrieves the page, it closes the connection to the server. The server then closes any tables that it may have opened, releases objects that were instantiated to generate the desired page and waits for another request. This is fundamentally different that many Foxpro developers will be accustomed to. When a user requests a subsequent page from your site, you will need to reopen any desired tables and reposition the table on the correct record.

Fox Trails takes care of many of these details for you, but you will need to ensure that you pass relevant fields, such as primary keys on subsequent requests. Review the Customer controller for specifics about how to simulate stateful programming in Fox Trails. You may also use ASP.Net session variables to maintain state. See the base controller class for a sample session variable. The SID property is passed to every controller class that inherits the base controller. You may use this property (it's a 36 character GUID) as a primary key in a shopping cart for instance, or create your own session variables as needed.

Making changes

Fox Trails was designed to provide an easier means of deploying changes to production sites. Since there is only a single COM object, it rarely needs changing, if at all. Once it's registered, most changes can be deployed by simply

copying the new files to the server. To deploy table changes to the server place empty tables in the Migrate directory and DO utility\migrate from the c:\foxtrails directory. See the source code for this utility for more details. It's very basic, please modify it to suit your needs.

Occasionally, a controller class will not be cleared from the program cache. This can happen is an error occurs or if control is redirected to another controller using the redirect() utility function. If you get a file in use error when attempting to copy a file, you'll need to stop the web server first. This will clear the Foxpro program cache and allow the new controller to be installed.

About the sample programs

Included with Fox Trails are a few sample controllers along with their corresponding layouts and views. These are intended to demonstrate the core features of the framework. Please take the time to understand how they operate before asking for assistance.

Getting help

You may contact me for assistance with Fox Trails via email: bmarquis65@mac.com, I will answer emails as timely as I can, but it may take a day or two for me to get back with you as I have a day job which demands a considerable amount of my time.

A P P E N D I X

The Dispatcher

How the Dispatcher works

The dispatcher is an OLEPUBLIC class that extends the Foxpro Session class. This provides an immediate benefit in that every web page requested from Fox Trails is given it's own data session. The dispatcher contains one primary method and a number of helper methods. The primary method is named "DispatchRequest" and it accepts two parameters that consist of the full URL and the second is a relative path to the ASP.Net page that called Fox Trails. You can see how it's called from ASP.Net by examining the WebRoot\Default.aspx page.

```
Dim dispatcher As New demo.Dispatcher
Response.Write(dispatcher.DispatchRequest(Request.Url.AbsoluteUri, Request.FilePath))
```

The dispatcher returns the html that Fox Trails generates which is then returned to the client's browser by ASP.Net.

Let's take a look at the flow of events within the Dispatcher

1. Create a DispatchContext object

The DispatchContext object is used to hold information about the user's request. This includes the Url that the user requested, the home directory of Fox Trails, the ASP.Net Request, Response and Session objects, the name of the controller to create, which method to call and the list of parameters to pass to that method. It also holds an errors array that you can use to report errors back to the user.

2. Parse URL

The second step in the DispatchRequest method is to read the incoming URL to determine where to send this request. This step identifies the controller to instantiate, the method to call on the controller and any parameters to pass to that method.

3. Create the controller instance.

4. Lookup the SID or Create a new one

This steps looks for an SID variable in either the ASP.Net Session or Request object. If it finds the variable, it's value is assigned to the controller's SID property. If the variable does not exist, create a new one.

5. Call the controller's OpenTables method.

The opentables method of the base controller class can be overridden to open any desired tables. This method can also be used to update the controller object from your database tables before the POST/GET variables are assigned.

6. Process the Query String

Step six takes any POST or GET variables and assigns them to properties of the controller instance.

7. Call the controller's AfterInit method.

This method can be overridden to do some processing before calling individual methods of the controller.

8. Call the desired method on the controller

9. Generate HTML to be returned to the client's browser.

There are two variables that are used to identify the View or Layout to use for the html generation.
oDispatchContext.RenderAction: defaults to the name of the method that was just called. Fox Trails looks in the Views\controller folder for a matching method.vml file.
oDispatchContext.RenderLayout: defaults to the name of the controller. Fox Trails looks for a Layouts\controller.vml file.
Either of these variables can be overridden to change the HTML being displayed. For instance, when calling "/Customer/Save", you may want to display a customer list after the current record has been saved.

Utility Programs

There are a number of utility programs in the Utility folder. These are designed to make your life easier.

UTILITY NAME	PURPOSE
etrim	Performs a trim and htmlencode, used in vml templates instead of a trim command to prevent cross-site scripting hacks.
fieldfor	Use this method to create an html input field for the specified database field.
fullurl	Returns the full url to the specified controller. For example: << fullurl("/customer/edit/"+customer.customerid) >>
htmlencode/htmldecode	Encodes/Decodes the input string with html encoding.
linkto	Creates an html hyperlink to the specified controller. <<linkto("/customer/edit/"+customer.customerid,customer.name)>>
redirect	Redirects program control to the specified controller.
rendervml	Evaluates the specified vml file. lcHtml = RenderVml("Views/customer/edit.vml",customer)
scaffold	Generates a CRUD maintenance set for the selected table.
urlencode/urldecode	Performs url encoding/decoding on the passed string.
webroot	Returns the base directory of FoxTrails on the web server.